On the Notion of Interestingness in Automated Mathematical Discovery

On The Notion Of Interestingness In Automated Mathematical Discovery

Simon Colton and Alan Bundy
Division of Informatics,
University of Edinburgh,
80 South Bridge,
Edinburgh EH1 1HN, Scotland.
simonco,bundy@dai.ed.ac.uk
http://www.dai.ed.ac.uk/~simonco

Toby Walsh
Department of
Computer Science,
University of Strathclyde,
Glasgow G1 1XH, Scotland.
tw@cs.strath.ac.uk

Abstract

We survey five mathematical discovery programs by looking in detail at the discovery processes they illustrate and the success they've had. We focus on how they estimate the interestingness of concepts and conjectures and extract some common notions about interestingness in automated mathematical discovery. We detail how empirical evidence is used to give plausibility to conjectures, and the different ways in which a result can be thought of as novel. We also look at the ways in which the programs assess how surprising and complex a conjecture statement is, and the different ways in which the applicability of a concept or conjecture is used. Finally, we note how a user can set tasks for the program to achieve and how this affects the calculation of interestingness. We conclude with some hints on the use of interestingness measures for future developers of discovery programs in mathematics.

1 Introduction

There has been some recent progress in surveying and extracting general principles of machine discovery in science, for example [18] and [37]. We aim to add to this by surveying five programs developed to perform discovery in mathematics. We restrict our discussion to programs whose main objective is to invent concept definitions and make conjectures in pure mathematics. This leaves out automated theorem provers (which discover proofs), and programs which discover mathematical results in other domains, such as the very important BACON programs, [19]. To compare and contrast the discovery programs, we detail what the aims of the project were, how the program worked and what contributions the programs made to mathematics and the understanding of mathematical discovery. We pay particular attention to the measures employed to estimate how interesting a concept or conjecture is.

Deciding whether something is interesting or not is of central importance in automated mathematical discovery, as it helps determine both the search space and search strategy for finding and evaluating concepts and conjectures.

Best-first searches using assessments of interestingness are often needed to effectively traverse large search spaces. When it becomes clearer what results are interesting, instead of just ignoring or discarding dull concepts and conjectures, the search space can be tailored to avoid some of them completely. Estimating interestingness is difficult because often it has to be done immediately after a concept or conjecture has been introduced, whereas the true interestingness of results and definitions in mathematics may only come to light much later.

In §4, we identify six reasons why a concept or conjecture might be considered interesting. We detail how the programs use empirical evidence to cut down on the number of false conjectures made. We show how the novelty of a conjecture can be determined by whether it, or an isomorphic conjecture, has been seen before, or whether it follows as an obvious corollary to a previous conjecture, and we detail the different ways in which a concept can be thought of as novel. We note that being surprising is a desirable property of conjectures and concepts and we show how programs will avoid making conjectures which are just instances of tautologies, and how they can assess the surprisingness of a conjecture or concept. We define the applicability of concepts and conjectures to be the subset of models to which they bear some relevance, and show that this measure can be used in a variety of ways. We also detail how programs can assess complexity and tailor their search strategies to find the most comprehensible results first. Finally, we look at how a user can set a program a particular task to achieve and how interestingness can be measured with respect to that task. By looking in detail at five discovery programs and extracting some common ways by which the interestingness of concepts and conjectures is estimated, we will be able to suggest possible ways for future programs to measure interestingness. We first discuss the scope of this paper and give some mathematical background.

1.1 Scope of the Paper

We approach the problem of interestingness in automated mathematical discovery pragmatically, by surveying five important programs which perform discovery tasks and extracting commonalities in how they estimate interestingness. It is not, therefore, in the scope of this paper to give psychological validation to the points we raise. Indeed, interestingness in automated mathematics is distinct in many ways to interestingness in mainstream mathematics, not least because an immediate assessment often has to be made in automated discovery, which is not usually the case in mainstream mathematics.

Measures of interestingness are used in many ways to facilitate machine creativity. In particular, we note that estimates of the worth of a concept or conjecture play a part in both the discovery aspects of machine creativity, as measures are used to drive heuristic searches, and the justification aspects, as measures are used to evaluate results after they have been produced. While we employ the notions of discovery and justification to help classify the measures used, it is not the aim of this paper to add to the philosophical discussion of these issues, and we suggest [17] or [29] for such a discussion. However, we will comment on the application of this study in automated mathematical discovery to the broader question of interestingness in automated scientific discovery. In particular, we will note in §6.1 that the measures set out in [37] by which humans can assess the output from scientific discovery programs are similar to the internal measures by which the programs assess interestingness.

1.2 Mathematical Background

Mathematical concepts include both the objects of interest in mathematics, for example even numbers, and the ways of describing those objects, for example being divisible by seven. The same concept can often be given as either an object or a description, the choice being dependent on the context. For example, in one context, someone might say that 'ten is an even number', whereas in another context it might be better to say that 'ten is even'. Usually, concepts have one or more definitions. For example, prime numbers are sometimes defined as: 'integers greater than 1 which are divisible by only 1 and themselves', and other times as: 'integers with exactly two divisors'. Often the definition will allow a test for membership, and sometimes the definition will enable examples (or models) of the concept to be generated. For example, the definition of prime numbers makes it possible to test whether a given number is prime, but gives no clue as to how to generate them, other than the generate and test method. Sometimes, it is difficult to find examples of a concept, and it is only possible to say that one exists, and other times, it is not even possible to do that. For example, it is not known whether there are any odd perfect numbers.[1]

Mathematical conjectures are statements about some concepts of interest. If an unrefuted argument is given which shows that the statement is true, the conjecture is referred to as a theorem, and the argument is called the proof of the theorem. However, if the truth of the statement remains in dispute, the statement is usually referred to as an open conjecture.[2] There are three common formats for conjectures discussed in this note. Firstly, it is often non-trivial to show that two definitions for a concept are actually equivalent, which gives rise to an **if-and-only-if** conjecture stating that an example satisfies the first definition if, and only if, it satisfies the second definition. **Implication** conjectures state that all objects of one type are also of another type and **non-existence** conjectures state that there are no examples of a particular concept.

A very brief overview of four domains, namely number theory, plane geometry, graph theory, and group theory, will suffice to appreciate how the programs discussed in this note work. Elementary number theory involves, amongst other things, the study of properties of numbers, relations between numbers and sequences of numbers. An important concept discussed here is the number of divisors of an integer, and the τ function calculates this value for a given integer. Plane geometry is the study of diagrams drawn on a flat surface involving points, lines, circles and other constructions. Important notions include lines being *parallel* if, no matter how far you extend them, they never cross, and a line being a *tangent* to a circle if it touches but doesn't cross the circle.

A simple graph is a set of nodes joined by undirected edges (see figure 1).

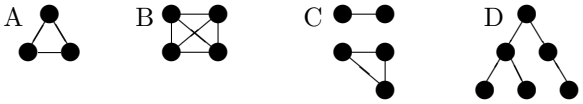


Figure 1: Four example graphs

[1] A perfect number is such that the sum of its divisors is twice the number itself.

[2] An exception to this rule was Fermat's last theorem, [34], which was called a theorem for over 350 years, even though no correct proof had been provided.

In elementary graph theory, two nodes are *adjacent* if connected by an edge, and the *degree* of a node is the number of edges the node is on. The graph theory concepts discussed here are (a) types of graph, such as complete graphs (where all nodes are joined by an edge as in graph A and B above), and (b) numerical values, for example the number of nodes or the maximum degree. An important area involves colouring the nodes of a graph. The *chromatic number* of a graph is the number of colours required to give it a 'proper' colouring, where no pair of adjacent nodes have the same colour. Paths are also important in graph theory, and the *radius* and *diameter* of a graph are related to the lengths of paths.

Finite algebras detail ways to take a pair of elements, a and b, from a finite set, and assign a third element, usually written $a*b$, to the pair. Each algebra has a different set of constraints, or axioms, which the assignments must satisfy. In finite group theory, there are three constraints, called the associativity, identity and inverse axioms, the details of which are not important here, apart from the fact that in groups there is always an *identity* element, id, for which $\forall a, a*id = id*a = a$. The assignment of $a*b$ is called multiplying a and b and groups are often presented with multiplication tables. For instance, figure 2 shows the multiplication tables for two groups with four elements.

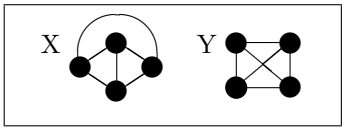
	a	b	c	d
a	a	b	c	d
b	b	c	d	a
c	c	d	a	b
d	d	a	b	c

	a	b	c	d
a	a	b	c	d
b	b	a	d	c
c	c	d	a	b
d	d	c	b	a

Figure 2: Multiplication tables for two groups of order 4

Elementary group theory concepts include relations between two elements, such as *commutativity,* where two elements a and b commute if $a*b = b*a$. Subgroups are subsets of elements which also form a group, and subgroup constructions are common, for example, taking the set of elements which commute with all the other elements gives a subgroup known as the *centre* of the group. Other elementary concepts include types of group, for example, if all pairs of elements in a group commute then the group is called *Abelian.*

A key notion in mathematics is *isomorphism,* where the same object can be represented in two or more ways. For example, in figure 3, graph X can easily be redrawn to look like Y. Graph theorists say that say that X and Y are isomorphic, ie. they are essentially the same. Similarly, group theorists say that groups A and B in figure 3 are isomorphic because swapping a and b in the body of table A gives table B. Often it is difficult to show that two objects are isomorphic, so properties are sought, called *invariants,* which are the same for any representation of the object. These can be used to show that two objects are not isomorphic, and various techniques, such as the one described in [27], can be employed to show that two objects are isomorphic.



A

	a	b
a	a	b
b	b	a

B

	a	b
a	b	a
b	a	b

Figure 3: Isomorphic graphs and groups

2 Machine Discovery Programs

The five programs we discuss in detail are the AM program which worked in elementary set and number theory, the GT program which worked in graph theory, the Graffiti program which is used in graph theory, the plane geometry system from Bagai et al, and the HR program which works in finite domains such as finite algebras, graph theory and number theory. For each one, we detail (i) the initial information given, (ii) the way in which concepts are represented, (iii) the techniques employed for inventing concepts, (iv) the techniques employed for spotting conjectures and (v) the successes of the project. The remainder of the discussions are designed to pay particular attention to the way in which the interestingness of concepts and conjectures is estimated by the program.

Other programs which performed discovery tasks in mathematics include Sim's IL program, [33], which invented operators on number types such as complex numbers. Also, Lenat's Eurisko program, [22], Haase's Cyrano programs, [13], Morales' DC program, [28], and Shen's ARE system, [32], all reconstructed or extended Lenat's original work on the AM system. The very recent SCOT program, [30], builds on the work of the ARE, Cyrano, GT and HR programs to perform specialised theory formation in graph theory. More specialised procedures have also been implemented by mathematicians for specific discovery tasks. For example, the Otter theorem prover has been used to discover new axiomatisations in algebra, [25], and the PSLQ algorithm, [2], has been used to identify a remarkably simple new formula for π:

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right).$$

2.1 The AM Program

The AM program, written by Douglas Lenat, performed concept formation and conjecture making in elementary set and number theory, as described in [7] and [21]. Starting with 115 elementary concepts such as sets and bags, AM would re-invent set theory concepts like subsets and disjoint sets, and number theory concepts such as prime numbers and highly composite numbers (with more divisors than any smaller integer). AM would also spot some well known conjectures, such as the fundamental theorem of arithmetic and Goldbach's conjecture - that every even number greater than 2 is the sum of two primes.

Concepts were given a frame representation with 25 facets to each frame, and none, one or multiple entries for each facet. Some of the facets were: (i) a definition for the concept (ii) an algorithm for the concept (iii) examples of the concept (iv) which other concepts it was a generalisation/specialisation of, and (v) conjectures involving the concept. AM repeatedly performed the task at the top of an agenda ordered in terms of the interestingness of the tasks. Each task involved performing an action on a facet of a concept. Usually the action was to fill in the facet, for example, find some other concepts which are specialisations of the concept or find some conjectures about the concept, but the action could also be to check the facet, eg. check that a conjecture was empirically true.

To perform a task, AM would look through its database of 242 heuristics, choose those which were appropriate to the task and perform each of the sub-tasks suggested by the chosen heuristics. Some sub-tasks detailed how to

perform the overall task at hand, but they were not limited to that. Some subtasks put new tasks on the agenda (which was how the agenda was increased). Some of the new tasks were to invent new concepts. When these were added to the agenda, AM would immediately create the frame for the new concept as knowledge present at the time was needed to fill in some of the facets of the concept. AM only filled in the information at this stage which took little computation, such as a definition and examples, and a task was put on the agenda to fill in each of the other facets of the newly formed concept.

Among the new concepts AM would suggest were: (i) specialisations, eg. a new function which was a previous one specialised to have equal inputs, (ii) generalisations (iii) extracted from the domain/range of a function, eg. those integers output by a function (iv) inverses of functions (v) compositions of two functions. Some tasks on the agenda were to find conjectures about a concept, including finding that (a) one concept was a specialisation of another (b) the domain/range of a concept was limited to a particular type of object or (c) no objects of a particular type existed.

Because there could be as many as 4000 tasks on the agenda at any one time, AM spent a lot of its time deciding which it should do first. Concepts, individual facets of the concepts and actions on the concepts were assigned numerical values indicating their worth. Whenever a heuristic added a task to the agenda, it would supply reasons, accompanied by appropriate numerical values why the action, concept or facet of the task was interesting. AM then employed a formula involving the number of reasons and a fixed weighted sum of the numerical values to calculate an overall worth for the task. The weighted sum gave more emphasis to the reasons why the concept was interesting than the reasons why the facet or action were interesting. When a heuristic was working out how interesting a concept was, it would collate and use another set of heuristics for the task. The heuristics used to measure the interestingness of any concept were recorded as heuristics 9 to 20 in [7], and included:

[9] A concept is interesting if there are some interesting conjectures about it.

[13] A concept is dull if, after several attempts, few examples have been found.

[15] A concept is interesting if all examples satisfy a rarely-satisfied predicate.

[20] A concept is more interesting if it has been derived in more than one way.

(Note that these have been paraphrased from Lenat's originals). AM also had ways to assess the interestingness of concepts formed in a particular way, for example the interestingness of concepts formed by composing two previous concepts could be measured by heuristics 179 to 189, one of which was:

[180] A composition $F = GoH$ is interesting if F has an interesting property not possessed by either G or H.

AM would also measure the interestingness of conjectures, so that it could correctly assess tasks relating to the conjectures facets of concepts. Heuristics 65 to 68 seem to be the only heuristics which do this, for example:

[66] Non-existence conjectures are interesting.

At any stage during a session, the user could interrupt AM to tell it that a particular concept was interesting by giving it a name. Lenat says in [7] that

users could "kick AM in one direction or another", and "the very best examples of AM in action were brought to full fruition only by a human developer". Many of AM's heuristics were designed to focus on chosen concepts, by spreading around the interest the user had shown in them. For example, these heuristics keep the attention on concepts and conjectures related to interesting concepts:

[16] A concept is interesting if it is closely related to a very interesting concept.

[65] A conjecture about concept X is interesting if X is very interesting.

In fact, AM could make a little interestingness go a long way: of the 43 heuristics designed to assess the interestingness of a concept, 33 of them involve passing on interestingness derived elsewhere. Therefore, if a user expressed an interest in a concept, the theory would develop around that concept.

There has been much debate about the AM program. In [31], Hanna and Ritchie were particularly critical of the methods AM used and the accuracy of Lenat's description of his work, and in [23], Lenat replied to this criticism. The main contribution of Lenat's work is an inspiration for how computers could do mathematics, ie. by creating concepts and conjectures of many different types and using heuristic methods such as analogy and symmetry to explore a domain.

2.2 The GT Program

The GT program by Susan Epstein performed concept formation, conjecture making and theorem proving in graph theory, as described in [9] and more fully in [10]. Given just the concept of a graph, GT would re-invent graph properties, such as being acyclic, connected, a star or a tree, (as shown in figure 4).

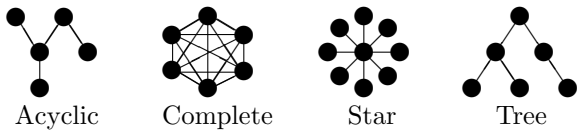


Figure 4: Graph properties re-invented by GT

Alternatively, given a set of user-defined and invented concepts describing graph properties, GT would make conjectures such as:

- A graph is a tree if and only if it is acyclic and connected.

GT successfully illustrated a possible mechanism for automated discovery in mathematics involving both deductive and inductive reasoning. This was possible because GT represented graph properties in a carefully thought out way developed in Epstein's PhD thesis, [8]. This representation was crucial as it allowed example generation, theorem proving and concept formation.

Each graph property was represented as a triple, $< f, S, \sigma >$, consisting of a set of base cases, S, a constructor, f, and a set of constraints for the constructor, σ, which together detailed the recursive construction of graphs from the base cases. For example, to define the star property above, the base cases would be just the trivial graph (with one vertex, no edges) and the constructor would add one vertex and an edge between the new vertex and an old vertex, subject to the single constraint that the old vertex must be on more edges than any

other vertex. Epstein was able to prove that 42 classically interesting graph theory concepts, including cycles, Eulerian graphs and k-coloured graphs, could be represented in this manner in a sound and complete way.

This representation could be used to generate examples of a concept (Epstein called this 'doodling') by starting with the base cases and repeatedly applying the constructor, subject to the constraints. Conjecture making and deduction was possible by spotting and proving that one graph property subsumed another (see [10]), or by showing that no graphs could have two particular properties. Concept formation was possible by: (a) *specialising* a previous concept by removing base cases, restricting the constructor, or strengthening the constraints, (b) *generalising* a previous concept by adding base cases, expanding the constructor, or by relaxing the constraints, or (c) *merging* properties A and B, for example creating a new graph property with A's base cases and constructor, but the constraints of both A and B, [subject to some conditions].

GT worked by repeatedly completing one of six types of project: (i) generate examples of graphs with certain properties, (ii) see if one property subsumed another (iii) see if two properties were equivalent, (iv) see if a merger between two properties would fail, (v) generalise a concept and (vi) specialise a concept. Each project was placed on an agenda following various rules:

- If a property has few examples in the database, then immediately generate more examples for it by 'doodling'.
- Two properties, P and Q, are better candidates for projects (ii) or (iii) above if the set of base cases for P and Q are similar. Two sets are most similar if they are equal, less similar if one is a subset of the other and less similar still if they only have a non-trivial intersection.
- Only perform specialisation or generalisation projects with a concept before doing conjecture-making projects if the concept is a 'focus' (see below).

As an overview, if a conjecture project was at the top of the agenda, before trying to prove the conjecture, GT would first see if there was empirical evidence against the conjecture, using the generated examples of the graphs [note that a conjecture was suggested only using the base cases]. If the project was to check a merger conjecture, then the merge step would take place, and only if no graphs of the merged type could be produced would an attempt be made to prove the conjecture. If a generalisation or specialisation project was at the top of the agenda, it would be carried out and some effort expended to generate examples of the new concept.

Focus concepts could be specified by the user if they were particularly interested in them, and, as well as restricting concept formation only to generalising, specialising and merging the focus concepts, GT would only make conjectures involving the focus concepts. If a concept was a generalisation of a focus concept, but no example graphs could be produced which were not examples of the focus concept, the new concept was discarded. Also, if only a few graphs could be generated with a newly formed property, the new concept was discarded.

By identifying the routine of ordering which conjectures to look at first, attempting to make and prove the conjectures, and performing concept formation only with the most interesting concepts, Epstein's implemented model of discovery successfully produced theories containing different kinds of conjecture and their proofs and concepts and graphs not present at the start of the session.

2.3 The Graffiti Program

The Graffiti program written by Siemion Fajtlowicz, makes conjectures of a numerical nature, mainly in graph theory, as described in [11], and more recently in [20]. Given a set of well known, interesting graph theory invariants, such as the diameter, independence number, rank or chromatic number, Graffiti uses a database of graphs to empirically check whether one sum of invariants is less than another sum of invariants. If a conjecture passes the empirical test and Fajtlowicz cannot prove it easily, it is recorded in the "writing on the wall" document, some of which is publicly available, [12]. Fajtlowicz also forwards promising conjectures to interested graph theorists. These types of conjecture are of substantial interest to graph theorists because (a) they often provide a significant challenge to resolve and (b) calculating invariants is often computationally expensive, so any bounds on their values are useful. As an example, the 18th conjecture in the writing on the wall states that, for any graph, G:

$$\begin{array}{c} \text{chromatic\_number}(G) \\ + \\ \text{radius}(G) \end{array} \leq \begin{array}{c} \text{max\_degree}(G) \\ + \\ \text{frequency\_of\_max\_degree}(G) \end{array}$$

Note that the only concept formation Graffiti undertakes is to add together two or more invariants, and the concepts are represented as fragments of executable code. The empirical check is time consuming, so Graffiti employs two techniques, called the *beagle* and *dalmation* heuristics, to discard certain trivial or weak conjectures before the empirical test:

The *beagle* heuristic discards many trivially obvious theorems, including those of the form $\mathrm{i}(G) \leq \mathrm{i}(G)+1$. (Note that invariants which are a previous invariant with the addition of a constant are used to make stronger bounds). The beagle heuristic uses a semantic tree of concepts to measure how close the left hand and right hand terms are in a conjecture, and rejects those where the sides are semantically very similar.

The *dalmation* heuristic checks that a conjecture says something more than those made by Graffiti previously. To use the dalmation test for a conjecture of the form $p(G) \leq q(G)$, Graffiti first collates all the conjectures it has ever made of the form $p(G) \leq r_i(G)$. Then, to pass the dalmation test, there must be at least one graph, G_0, in Graffiti's database which for all the r_i, $q(G_0) \leq r_i(G_0)$. This means that, for at least one graph, $q(G)$ gives a stronger bound for $p(G)$ than any invariant suggested by a previous conjecture, so the present conjecture does indeed say something new about Graffiti's graphs.

Another efficiency improving technique employed by Graffiti is to restrict the database of graphs to only those which are a counterexample to a previous conjecture. A third efficiency technique is to remove by hand any previous conjectures which are subsumed by a new conjecture. For example, Fajtlowicz would move the old conjecture $i(G) \leq j(G) + k(G)$ to a secondary database, if the conjecture $i(G) \leq j(G)$ was made. However, if the latter conjecture was subsequently disproved, the former conjecture would be restored.

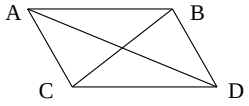
As Fajtlowicz adds concepts to Graffiti's database, the writing on the wall reflects the new input, eg. conjectures 73 to 90 involve the coordinates of a graph. Fajtlowicz can also direct Graffiti's search by specifying a particular

type of graph he is interested in. For example, conjectures 43 to 62 are about regular graphs. To enable this kind of direction, Fajtlowicz informs Graffiti of the classification of its graphs, into, say, regular and non-regular graphs. Then, if Graffiti bases its conjectures on only the empirical evidence supplied by the regular graphs, the conjectures will only be about those graphs. To stop Graffiti re-making all of its previous conjectures, the *echo* heuristic uses semantic information about which graph types are a subset of which others, and rejects conjectures about the chosen type of graph if there is a superset of graphs for which the conjecture is also true.

In terms of adding to mathematical knowledge, the Graffiti program has been extremely successful. Its conjectures have attracted the attention of scores of mathematicians, including many luminaries from the world of graph theory. There are over 60 graph theory papers which investigate Graffiti's conjectures. While Graffiti owes some of its success to the fact that the inequality conjectures it makes are of a difficult and important type, this should not detract from the simplicity and applicability of the methods and heuristics it uses.

2.4 Bagai et al's System

The discovery program developed by Rajiv Bagai et al, described in [1], worked in plane geometry by constructing idealised diagrams and proving theorems stating that certain diagrams could not be drawn. Each concept consisted of a set of first order statements representing a diagram in plane geometry. The diagrams involved points and lines and relations between the points and lines, such as a point being on a line or two lines being parallel. For example, a parallelogram and its diagonals, as in figure 5 below (taken from [1]), could be described by stating that there were four ingredient points, A, B, C and D, six lines (one between each pair of distinct points) and two relations, namely that lines AB and CD were parallel and that lines AC and BD were parallel.



Ingredient points:	A,B,C,D
Relationships:	parallel(line(A,B),line(C,D))
	parallel(line(A,C),line(B,D))

Figure 5: A parallelogram and diagonals, and its representation

The system required no initial information, and starting with an empty set, concepts like the parallelogram were made by adding new ingredient points and new relations to a previous concept. Each time a new relation was added, a conjecture was made that the resulting concept was inconsistent, ie. that it was not possible to draw the diagram. To prove the conjecture, the concept was turned into a collection of polynomials and inequalities which were passed to an efficient theorem prover, [4]. If the theorem was proved, then the concept was discarded, and the theorem was recorded and output. If the theorem was not proved, the concept was kept and used to build new concepts from.

Many methods were employed to reduce the number of times the system used the theorem prover. Firstly, as previously stated, only consistent concepts were built upon, as a concept which was an extension of an inconsistent concept would itself be inconsistent. By also restricting to only adding one relation at a time, if the concept produced was inconsistent, the additional relation must have caused the inconsistency. This enabled better presentation of the theorems. For

example, if the relationship: `parallel(line(A,D),line(B,C))` was added to the parallelogram concept above, this would produce an inconsistent concept. As the inconsistency was caused by the new relation, instead of just stating that parallelograms with parallel diagonals cannot be drawn, the system could state that: "Given a parallelogram, the diagonals cannot be parallel." (Noting that we've substituted the word parallelogram for the first order description involving parallel lines).

Another way to reduce the time spent using the theorem prover was to avoid proving the inconsistency of a concept which was isomorphic to a previous one. Two concepts were isomorphic if a permutation of the ingredient points of the first produced the second. To get around this problem, whenever a concept was introduced, all of its isomorphic concepts were also built, so that they could be recognised and ignored if re-constructed by a different route later on. Also, to cut down on the occurrences of later theorems which implied earlier ones, a breadth first search was used where a step could only be the addition of either a single ingredient point or a single new relation. This meant that the most general diagrams were constructed before the more specific ones and therefore the most general versions of theorems were produced first. Not only could the program re-discover well known results such as Euclid's 5th postulate, it also provides a very clear and concise theory for the automatic production of a subset of plane geometry concepts and a set of theorems about the non-existence of models for certain concepts.

2.5 The HR Program

The HR program by Colton et al, as described in [6], was originally developed to perform concept formation in finite group theory, but the methods applied to many finite domains, and HR has been used in many different finite algebras, as well as number theory and graph theory. Starting with just the axioms of group theory, HR can re-invent classically interesting concepts such as centres of groups, Abelian and cyclic groups and orders of elements. HR works directly with the models of concepts (stored as data-tables), and constructs new concepts by taking the data-tables of old concepts and manipulating them using one of ten production rules to produce a new data-table. The production rules include ways to specialise and generalise tables, and ways to combine tables and find the compliments of tables (ie. the data which is *not* in the table). From information about how a concept was constructed, HR can generate a definition for the concept whenever one is needed.

HR encounters a combinatorial explosion because a single concept can often be transformed into around 20 new ones, and any pair of concepts can be combined into a third. A heuristic search is used which chooses the best concept to use in each concept formation step. HR has a variety of ways to measure concepts and a weighted sum of measures is taken to indicate an overall level of interestingness for the concept. The weights are set by the user and depend on the nature of the concepts they are looking for. One way to use HR is to supply a 'gold standard' categorisation of the groups it has, and ask HR to find a function, the output of which will categorise the groups correctly (groups with the same output are put in the same category). HR can then measure how close each concept gets to the gold standard. For example, given the isomorphic classification of the groups up to order 6 (see §1.2) HR found this function which

achieves the correct categorisation:

$$f(G) = |\{(a, b, c) \in G^3 : a * b = c \ \& \ b * c = a\}|.$$

If the user has no particular task in mind, they can ask HR to explore the domain. HR has certain measures which indicate desirable properties of a concept, and users can stress some of these if they wish. The *parsimony* measure of a concept is inversely proportional to the size of the data-table for the concept. The data in a table corresponding to a particular group can be used to describe that group, and so a small table is advantageous as this means more parsimonious descriptions. HR can also assess the *novelty* of a concept, which is inversely proportional to the number of times the categorisation produced by the concept has been seen already, (with more unusual categorisations being more interesting). Finally, HR can measure the *complexity* of a concept which is inversely proportional to the number of old concepts appearing in its construction path. This gives a rough indication of how complicated the definition of the concept will be, and more concise definitions are desirable.

HR can make if-and-only-if conjectures by spotting that the data-table of a newly formed concept is exactly the same as a previous concept, and conjecturing that the concepts are equivalent. When this happens, definitions for each concept are generated and used to write the conjecture in a way acceptable to the Otter theorem prover, [24], which HR asks to prove the conjecture. For example, when HR invents the concept of elements, a, for which $a * a = a$, it spots that the new data-table is the same as the one it has for the concept of the identity element, id, and the following conjecture is generated:

$$\forall a, (a = id \iff a * a = a).$$

This is broken into $(a = id \longrightarrow a * a = a)$ and $(a * a = a \longrightarrow a = id)$, which are both passed to and easily proved by Otter. Before passing a conjecture to Otter, HR uses some simple deductive techniques to check whether the conjecture follows easily from those already proved. HR can also make implication and non-existence conjectures using the empirical evidence from the data-tables.

HR has a set of 'sleeping concepts', such as the trivial group, and when a concept is conjectured to be the same as these, the conjecture is flagged so that the user can pay special attention to it (or choose to ignore it). Conjectures are assessed in two ways. Firstly, the *surprisingness* of a conjecture measures how different the two (possibly) equivalent concepts are, by evaluating the proportion of concepts which appear in the construction path of one but not both of the concepts. This gives some indication of how different looking the definitions of the equivalent concepts are going to be. Secondly, if a conjecture is proved, Otter will provide a *proof length* measure in its output, which gives some indication of the difficulty of the proof. A cycle of mathematical activity is closed by HR because it uses the assessments of conjectures to assess the concepts discussed in the conjectures, thus advancing the concept formation.

If the equivalence of two definitions is proved, HR uses this fact to reassess the concepts involved, and keeps only the least complex definition for the concept. If Otter cannot prove a conjecture, HR passes it to the MACE model generator, [26], which is asked to find a single counterexample to the conjecture. If MACE is successful, the counterexample is added to HR's database and all previous concepts and measures are re-calculated, giving HR a

better idea of the theory it is exploring. All future conjectures will be based on the additional data provided by the new example.

HR's biggest success so far has come in number theory, where it has identified 14 interesting sequences missing from the encyclopedia of integer sequences.[3] One of these sequences was originally defined as recently as 1990, [15], but the others are believed to be new to mathematics. HR has also found some interesting theorems about these sequences, [5], for example, if the sum of the divisors of an integer is prime, then the number of divisors must be prime.

3 A Comparison of the Programs

A detailed comparison of the measures of interestingness used by the programs is given in §4, and we restrict ourselves here to a comparison of some logistical aspects of the programs. We first focus on the role of the user and follow this with a table for comparing the programs.

3.1 The Role of the User

It is important to determine what role the user played in guiding the search for concepts and conjectures in these programs, as this had an impact on how the program measured interestingness. Firstly, in many cases, the user could set tasks for the concepts/conjectures to achieve. We discuss this fully in §4.6, as it is more appropriate to think of achieving tasks as a measure of the interestingness of a concept or conjecture.

With the reporting of AM following Lenat's thesis, it is easy to forget that AM was designed to be interactive. With each concept stored as a frame containing much information, and because AM started with 115 concepts, it was only possible to invent around 170 concepts. In order to find interesting concepts with such a short search, AM relied heavily on input from the user, who could stop AM at any time and express an interest in a concept by giving it a name. The heuristics were designed so that a sizeable amount of the search would revolve around the chosen concept. Hence, if Lenat expressed an interest in, say, the division concept, this greatly increased the chances of AM finding the τ function (number of divisors), prime numbers (exactly two divisors), and so on. Thus the user played a large part in shaping AM's search.

The user could also have an impact on the searches made by the GT program. By identifying a focus concept, the search would centre around that concept. However, GT functioned perfectly well with no user intervention. Perhaps the biggest role of the user in the Graffiti program is to supply the many interesting graph theory concepts that it uses to make conjectures. Also, Fajtlowicz maintains the database of previous conjectures by pruning any which are subsumed by more general ones, and any which are trivial to prove. Finally, as in the GT program, the user can restrict Graffiti's search to only involve certain concepts, thus focusing the search around those concepts. The user seemed to play no part in the search undertaken by the Bagai et al program, except perhaps to tell it how many theorems were required. Finally, the user can set certain parameters before running HR, in an attempt to guide the search in advance. Once a session has started, the user will not interrupt or alter the search in any way.

[3] See `http://www.research.att.com/~njas/sequences`.

The output of discovery programs is usually intended for a human audience, so enabling the user to guide the search in some way is a good idea. The impact of the user on the theory produced can depend on the size of the search space and various limitations of the program. Note that only the Graffiti and HR projects are current, and only these have added to mathematics, with Graffiti providing many conjectures in graph theory, and HR contributing to number theory. Only Graffiti was developed and used by mathematicians, with the others developed as artificial intelligence projects designed to illuminate and model aspects of discovery in mathematics, rather than as collaborators with mathematicians, an important distinction highlighted in [37].

3.2 Comparison Table

This table is provided for quick comparison of the programs, and details in the table can be expanded by reference to the relevant subsection of §2.

Program	Year	Domains	Representation of concepts	Initial information
AM	1976	set, number	frames	115 concepts
GT	1987	graph	base case, constructor and constraints	a few concepts
Graffiti	1988	graph, number, geometry	code fragments	many interesting concepts
Bagai et al	1993	geometry	first order statements	nothing
HR	1997	finite algebras, number, graph	data-tables	axioms (eg. 3 in group theory)

Program	Concept Formation Techniques	Conjecture Types	Inference Mechanisms	Addition to Mathematics
AM	generalise, specialise, compose	if-and-only-if, implies, non-exists	induction, model generation	-
GT	generalise, specialise, compose	if-and-only-if, implies, non-exists	induction, deduction, model generation	-
Graffiti	addition of invariants	inequalities	induction,	new graph theory theorems
Bagai et al	adding relations and objects	non-exists	induction, deduction	-
HR	generalise, specialise, compose, compliment	if-and-only-if, implies, non-exists	induction, deduction, model generation	new number theory concepts and conjectures

Perhaps the most striking difference between these programs is the different representations for concepts used in each program. The choice may be because it facilitates the production of models (GT, Graffiti) or the proving of theorems (GT, Bagai et al) or for efficiency reasons (HR).

4 Assessing Interestingness

We cannot discuss measures of interestingness without addressing how the measures are used. For example, one program might say that the concept of even prime numbers is interesting because a conjecture can be made that 2 is the only one, whereas another program might say that they are dull because it can only find one example. Here, the same measure has been used (see §4.4 below), but different conclusions have been drawn. Therefore we have to clearly separate the measures for interestingness from their uses. One common use of interestingness is to improve the efficiency of the programs. To save time checking and proving conjectures, some of them are discarded before even checking them empirically, and the reason to perform an empirical check is, of course, to cut down on the time spent trying to prove false conjectures.

Another common use of interestingness is to improve the appeal of the output. It is not possible to avoid all uninteresting concepts or conjectures when constructing a theory and interestingness measures can be used to filter the output depending on the user's needs. Also, measures of interestingness can guide the search so that the program can make informed progress into the space and find interesting concepts that it might take a longer time to find with an exhaustive search. A more specific use of interestingness measures is to predict in advance how difficult a conjecture will be to prove, which, in all but some trivial circumstances is not easy to do. Finally, interestingness measures can be used to steer the concept formation towards a particular concept which performs a user-defined task. Having identified some uses for interestingness, we can detail certain general types of measures and look at how each one is used.

4.1 Empirical Plausibility of Conjectures

A conjecture is likely to be uninteresting if the empirical evidence a program has provides counterexamples. This does not mean that false conjectures in general are uninteresting, as the production of counterexamples is a worthwhile pursuit. However, if a counterexample is found using the data a program has, the conjecture cannot provide this pursuit. Only the system developed by Bagai et al makes conjectures which have not been first verified by some empirical evidence. In this case, the efficiency and power of the theorem prover and the nature of the idealised geometrical domain make it unproductive to look for counterexamples. The AM program is the only one which doesn't immediately discard a conjecture proved false by empirical evidence. In this case, an attempt is made to alter the conjecture to make it fit the data. One way to do this is to exclude what Lenat calls 'boundary' integers, so for example, the conjecture 'all primes are odd' becomes the conjecture 'all primes except 2 are odd'.

HR and Graffiti use all of their data at once. HR uses its data to spot a conjecture, so by the time the conjecture has been made, the empirical check has been completely performed. Similarly, once a conjecture has been suggested to Graffiti, all the empirical evidence is used to check it. Note that both these programs keep the amount of empirical data down to a minimum because they only store models which have been generated as counterexamples to previous conjectures. GT employs a more efficient system because a conjecture is suggested by the small amount of empirical evidence in the set of base cases, and only those conjectures passing this test are checked against all the examples for

the concepts. Similarly, AM will make a conjecture based on a little empirical evidence, then try to generate more models to disprove the conjecture.

4.2 Novelty

Because of the redundancy often inherent in searches for concepts and conjectures, it is important to be able to spot when a repetition has occurred. Each program either tailors its search to reduce repetitions, or can spot them when they occur. Thus they measure the novelty of a concept or conjecture statement, and reject those which have been seen already. The Graffiti program goes to the length of storing conjectures between sessions. Another issue of novelty in programs searching for conjectures is whether a theorem follows as an obvious corollary to a stronger theorem, in which case the weaker result does not say anything particularly new. Graffiti works hard to show that a new conjecture says something more than the previous ones, by checking that there is at least one graph for which the inequality in the conjecture is stronger than all the previous ones (the dalmation heuristic), and by checking that the conjecture is not implied by previous ones (the echo heuristic). Spotting the implication of one conjecture by another is also used in Graffiti to improve efficiency: if a later conjecture turns out to be stronger than a previous one, the earlier one is removed, hence saving Graffiti time when looking through old conjectures.

The HR program deals with the implication of one conjecture by previous ones by attempting to prove all stronger conjectures than the one it is considering. For example, if interested in the conjecture $P \ \& \ R \rightarrow Q$, HR first uses some simple deductive techniques to see if it follows as a corollary to its previously proved results. If not, HR tries to prove $P \rightarrow Q$, and $R \rightarrow Q$ and if either turns out to be true, the original conjecture is discarded and the stronger result is kept. The most efficient way to deal with one conjecture implying another is to tailor the search to produce the most general conjectures first, reducing the chance that later conjectures will imply earlier ones. This technique is used in the system from Bagai et al, but they point out in [1] that it is still possible to produce a conjecture which implies an earlier one.

A concept can be shown to be novel with empirical evidence. For example, if a function produces some output for a given input that no other function produces, it must be novel. Given only a limited amount of data though, it can be difficult to tell whether two concepts are different. Bagai et al's system (which uses no data at all) tackles this by generating all possible isomorphic concepts whenever a new concept is introduced, so that if an isomorphic concept to the new one is reached by another route, the system will spot this. HR's conjecture making abilities rely on the fact that a proof is often needed to tell that two concept definitions are in fact equivalent, and, like AM, HR assesses a concept as more interesting if it has multiple definitions. If a concept isn't the same as one already in the theory, it is possible to assess how much it differs from the others, using certain properties of it. AM, for example, gave extra interestingness to newly formed concepts, ie. those with the novel property of being recently invented. The HR program assesses the novelty of a concept by the novelty of the categorisation of groups it gives. It is important to make the distinction between one concept having two properties (eg. two definitions), which is often interesting, and two concepts sharing the same property (eg. a categorisation), which often detracts from the interestingness of both.

4.3 Surprisingness

As portrayed in [12], when asked what makes a good conjecture, the mathematician John Conway said without any hesitation: "it should be outrageous". This is good advice, and in some cases, an assessment of how surprising a conjecture is can be made automatically. The least surprising conjectures are those which are just instances of tautologies. For example, given objects of any type, A, and a predicate of any nature, p, the conjecture

$$\forall A,\ not(not(p(A))) \iff p(A)$$

is always going to be true, and conjecture finding programs should avoid making these and similar conjectures.

To avoid making tautologies of a particular type, GT did not attempt subsumption conjectures where one of the concepts was a specialisation of the other. The HR program avoids certain tautologies by forbidding particular series of concept formation steps, eg. not allowing two negation steps in succession. Graffiti uses a semantic tree to measure how much invariants i and j differ in the conjecture: $\forall$ graphs G, $i(G) \leq j(G)$, and the beagle heuristic discards many tautology conjectures which involve very similar concepts, such as $i(G) \leq i(G) + 1$. Whereas Graffiti uses surprisingness only to discard conjectures, when the HR program makes a conjecture that two concept definitions are equivalent, it has semantic information about those concepts, so can tell how different they are, giving an indication of how surprising the conjecture is. HR uses the heuristic that concepts appearing in surprising conjectures are more interesting, which helps drive a best first search. While HR and Graffiti can estimate the surprisingness of conjectures, only AM measured the surprisingness of a concept: it gave extra interestingness to concepts which possessed a property not possessed by its parents (see heuristic 180 in [7], for example).

4.4 Applicability

The applicability of a predicate can be defined as the proportion of models in a program's database which satisfy the predicate. Similarly, the applicability of a function can be defined as the proportion of models in a program's database which are in the domain of the function. This measure is somewhat analogous to the empirical plausibility of conjectures, but can itself be extended to cover conjectures: the applicability of a conjecture can be defined as the proportion of models in a program's database which satisfy the conjecture's preconditions. These measures are used in a variety of ways as follows.

In AM and GT, if a newly formed concept had low applicability (ie. few examples), a task was put on the agenda to generate some more models that it applied to. If a concerted effort to generate examples still resulted in a low applicability, GT would discard the concept as uninteresting, and AM would give it a low interestingness score. In the special case where the applicability was zero, (ie. no examples were found), both GT and AM would make the conjecture that none exist. The HR program makes similar non-existence conjectures, and other conjectures about the applicability of a concept, for example, that it is restricted to the trivial group. In Bagai et al's system, the whole point was to prove that certain concepts have no models (ie. the diagrams cannot be drawn), which is equivalent to showing that the applicability of a concept is zero. So we

see that concepts with little or no applicability are often thought of as dull, but the conjecture that this is true is interesting.

Furthermore, in GT, if a generalisation step produced a concept with no greater applicability than the one it generalised, or if a specialisation step produced a new concept with a greater applicability than the one it was supposed to specialise, the new concept was discarded. In Graffiti, if the user has specified an interest in a particular set of graphs (ie. those with a particular quality), then if a conjecture is output which is applicable to a superset of that set, it is discarded as being too general (the echo heuristic). Also, in HR, the parsimony measure prefers concepts with small data-tables, and hence small applicability. This gives an emphasis to specialisation procedures and can be useful in controlling the search. Finally, the AM program used 'rarely satisfied predicates' - those with low applicability - to assess other concepts. For example, heuristic 15 from [7] gave more interestingness to functions whose output always satisfied one of the rarely satisfied predicates that AM had come across. We see that applicability is a common measure, but how it is used is determined by the context in which the discovery task is attempted.

4.5 Comprehensibility and Complexity

As the programs are intended to produce output for a user to read, more comprehensible concepts and conjectures are usually of more interest. The GT and Bagai et al programs constructed concepts incrementally so that the most comprehensible ones were introduced first. The HR program employs the complexity measure which prefers concepts with smaller construction paths (which roughly relates to how complicated their definition will be). HR's best first search is not guaranteed to produce the least complex concepts first, so, if two concepts are proved to be equivalent, the least complex definition of the two is kept. Also in HR, concepts are used to describe objects in the theory, and the parsimony measure prefers concepts giving more concise descriptions. Also, by naming AM's concepts, the user helped make the theory more understandable.

The comprehensibility of concepts gives one, albeit shallow, indication of their complexity, and usually simpler concepts are desirable as these are often more understandable, as discussed in [16]. An alternative way to assess the complexity of a concept is to evaluate how much is known about it. AM counts how many conjectures there are involving a concept and uses this as a measure of the interestingness of the concept. The HR program goes one stage further and assesses not only the conjectures but also the proofs of them, and uses this to measure the interestingness of the concepts involved in the conjectures.

Novelty and surprisingness measures often prune trivial conjectures, as tautologies, conjectures following as obvious corollaries to previous theorems and those which are unsurprising have a greater likelihood of being easy to prove. Removing these conjectures should increase the average difficulty to prove the conjectures remaining. However, this is separate from the issue of how difficult a conjecture is to understand. Preferring conjectures about less complex concepts will increase the overall comprehensibility of the theory, and choosing a simple format for conjectures can also help. For example, as noted in [37], it is easy to understand Graffiti's inequality conjectures. Further, the program from Bagai et al presents its theorems not as unsatisfiability results, but as relations which cannot occur once a diagram has been set up, a more understandable format.

4.6 Utility

There are ways by which the user can explicitly express interest in particular concepts or conjectures. This is clear in the AM program which was designed as an interactive program where the user can interrupt the session at any time and express an interest in particular concepts. AM's heuristics were set up to pay particular attention to these concepts, and the limited number of concepts AM could produce in a session were heavily biased by the user's choice. Here we see that the user wanted AM's concepts and conjectures to perform a particular task, namely to discuss the concept chosen by the user. This was taken a stage further in the GT and Graffiti programs, where the user could specify a 'focus' concept, indicating that only conjectures involving the chosen concept were to be produced. In GT's case, this also meant that concept formation was to be limited to specialising and generalising focus concepts.

In the Graffiti program, all the proved conjectures give a bound for an invariant (which may save computation time), so the search space has been designed with a task in mind. Also, by giving a 'gold standard' classification of groups to HR, users are expressing an interest in concepts which achieve that categorisation. By giving concepts and conjectures particular tasks to achieve, a program can measure how close each comes to completing the tasks and use this to estimate interestingness, which will hopefully drive the best-first search towards something which achieves the task.

5 Classifying Interestingness Measures

A natural progression for a measure of interestingness is to turn from a justification measure to a discovery measure as a discovery program evolves. For example, the complexity measure employed by HR was initially used to prune the output, so the user only saw the more comprehensible results. Later versions of HR incorporated the measure into the heuristic search, so that only the comprehensible results found during the search were built on to produce more results. At present, the complexity measure is more often used to tailor the search space, in effect making the search depth limited and guaranteeing that only comprehensible concepts are produced.

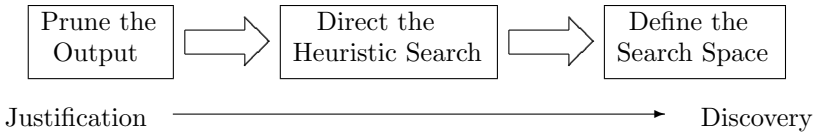


Figure 6: Possible progression of an interestingness measure

We see that an interestingness measure can be classified as either *pruning, directing* or *defining* the search space. It is often the case that pruning and directing measures are those which are difficult to turn into defining measures. For example, it is difficult to tailor the search to produce only surprising conjectures, so HR uses surprisingness only as a pruning measure. Other examples of pruning measures include the way in which the AM, GT and Graffiti programs discard conjectures which fail empirical tests, and the way in which GT discards newly formed concepts which did not achieve the desired specialisation or gen-

eralisation. Examples of directing measures include the utility measure in AM (ie. the user intervening to point to the interesting concepts), the parsimony measure in HR, and GT's choice for concepts to try in subsumption and equivalence tasks. Examples of defining measures include HR's complexity measure, the setting of focus concepts in GT and Graffiti and the way the Bagai et al system only built on consistent concepts. Also, the representation of concepts in the GT program was influenced by the fact that interesting graph types can be expressed by base cases, constructors and constraints.

Another way in which we can describe an interestingness measure is as *immediate* or *evolving*. Immediate measures perform a calculation involving some aspects of the concept or conjecture and produce a value which is assigned to the concept and never changes. Examples of immediate measures include the surprisingness and complexity measures in HR and the beagle and echo heuristics in Graffiti. While immediate measures are necessary to give some initial idea of the worth of concepts and conjectures, such snapshot evaluations can often be fairly blunt. Evolving measures, in contrast, are constantly updated as the theory expands. This expansion might be in terms of new models introduced into the theory. For example, HR's parsimony, novelty, invariance and discrimination measures are all re-calculated when a new example is introduced as a counterexample to a conjecture. Likewise, when new models were introduced to AM and GT, all the applicability measures were re-calculated. The expansion of the theory can also be in terms of the concepts and conjectures added. For example, categorisations in HR become less novel as more concepts producing that categorisation are added, so novelty is an evolving measure. Similarly, if a new conjecture is spotted by AM or HR, the interestingness of the concepts involved in the conjecture is increased.

6 Conclusions

Assessing the interestingness of a concept or conjecture automatically is difficult because a program must try to predict how much useful mathematics will result from an investigation of the concept or the attempts to prove the conjecture. Fermat's last theorem, for example, could easily have been relegated to the appendix of a number theory text if it had not been so difficult to prove. Also, as in discovery of any kind, it is often necessary to have expert knowledge to decide whether an invention has any far reaching implications or applications, as pointed out in [18]. By comparing and contrasting five machine discovery programs, all of which guide their search towards more interesting results by making decisions about the possible interestingness of those results, we have noted that the programs use plausibility, novelty, surprisingness, applicability, comprehensibility and utility to estimate the interestingness of the concepts and conjectures they produce. We summarise these methods in §6.2 and §6.3 in the form of hints for future developers of mathematical discovery programs.

6.1 Interestingness in Other Sciences

The abstract nature of the objects discussed in pure mathematics domains lends itself to automated discovery. Because the objects are abstract, they are often straightforward to represent in a program. This allows no room for experimental

error, and so a conjecture is usually not interesting if it is false in a few cases. This means that, as for all the programs mentioned here, no statistical measures are needed to assess the worth or plausibility of a conjecture. So, whereas a result in the physical sciences which was true for 90% of the data would be noteworthy, it would probably be uninteresting in pure mathematics. For this reason, and because the measures employed in scientific discovery programs such as ARROWSMITH, [35] and MECHEM, [36], are often quite domain specific, it is difficult to assess the measures in the terms discussed here. Indeed, in genetic approaches to automated scientific discovery, such as that discussed in [3], a fitness function is developed for each individual problem addressed, and the interestingness of possible solutions is measured against that function.

As discussed recently in [14] and [38], the question of interestingness is a key issue in machine learning and in particular knowledge discovery in databases. It is clear that the general methods highlighted here for mathematical discovery programs to assess their results can be applied to other scientific domains. For example, in the MECHEM program, as with some of the systems discussed above, simplicity is a defining measure of interestingness, as the search is carried out in stages which minimise the length of the solution produced (which in MECHEM's case is a reaction mechanism in chemistry).

Finally, we note that the four measures Valdés-Pérez suggests in [37] for assessing the output of a discovery program are novelty, plausibility, intelligibility and interestingness. The first three of these are identified here as internal measures used by some of the above programs, and the fourth is the subject of this paper. It should come as no surprise that internal measures for the interestingness of results often follow the same general principles as the measures used to assess the output from discovery programs. It is clearly a good strategy to identify which interesting initial results are output from a program and implement measures to increase the yield of such concepts.

6.2 The Interestingness of Conjectures

In summary, it is often a good idea to prescribe a definite task for a concept or conjecture to achieve, and design measures of interestingness around this. If this is not possible, and an estimate of the interestingness of a conjecture is going to be made, some of the following points could be taken into consideration:

- The conjecture should be empirically true.

This can be achieved by only making conjectures backed up by all available data, or suggesting a conjecture by some other means and then using all the data to discard the conjecture if necessary, or altering a conjecture so that any data which disproves it is no longer applicable.

- The conjecture should be novel with respect to previous ones.

This can be achieved by understanding and checking how two conjectures can be equal or isomorphic in the domain of interest. Also, conjectures which are implied by previous, stronger conjectures, should either be avoided by tailoring the search space, or rejected when found.

- The conjecture should be surprising in some way.

This can be achieved by avoiding or discarding well known tautologies, and by using information about the concepts discussed in the conjecture to estimate how unlikely the suggested relation between them is.

- The conjecture should discuss some non-trivial models.

This can be achieved by discarding conjectures where the set of models satisfying the preconditions is trivial or uninteresting. Note that in certain circumstances, this kind of highly specialised conjecture may actually be of interest to the user.

- The conjecture should be understandable, but non-trivial to prove.

This can be achieved by assessing the quantity and diversity of concepts involved in a conjecture and removing overly complicated conjectures, or by fixing the search strategy to output the simplest conjectures first. Making conjectures in a well known format will help understandability. If the conjecture has been proved, the proof could be used to estimate how difficult the theorem was.

6.3 The Interestingness of Concepts

If an estimate of the interestingness of a concept is going to be made, some of the following points could be taken into consideration:

- The concept should have models.

This can be achieved by checking available data for models which satisfy a predicate or are in the domain of a function. If no models exist, an effort should be made to generate some. It may be necessary to prove that no models exist, and discard the concept (but keep the theorem).

- The concept should be novel with respect to previous ones.

To achieve this, the search should ensure that no two obviously isomorphic definitions can be made, and avoid paths which will ultimately lead to the same concepts. Then, if the models of one concept are the same as another, the concepts are possibly equivalent, which should lead to a proof of this fact. If a concept is indeed semantically different to all the others, then the novelty of various properties (such as the way it categorises models) could be assessed.

- There should be some possibly true conjectures made about the concept.

By the qualification of possibly true conjectures, we note that while false conjectures about the nature of a concept are usually of no interest, an open conjecture can often be more interesting than a proved theorem.

- The concept should be understandable.

This can be achieved by designing the search to construct concepts with the simplest definitions first, and by keeping the simplest definition when it has been proved that two definitions for a concept are equivalent.

- The concept should have a surprising property.

A surprising property may be something that isn't true of the parent concepts.

Building on the techniques for automated discovery that have been developed in artificial intelligence and cognitive science, and learning from the results of programs developed in mathematics, an effort can be made to write more programs which act as collaborators with working mathematicians. The production of intelligently suggested conjectures and concepts plays an integral and important part in developing a mathematical theory. Automating these processes is a worthy area for research which is beginning to be recognised by the automated deduction community, [39]. How programs estimate the interestingness of the concepts and conjectures they produce is central to building intelligent discovery programs, and we hope that the notions of interestingness derived here will be of some help to future developers.

Acknowledgements

We would like to thank Pat Langley and Raúl Valdés-Pérez for sending us survey papers on machine discovery programs. We would also like to thank Susan Epstein for her comments explaining aspects of the GT program, Jan Żytkow for his comments on the system by Bagai et al and Craig Larson for helping us understand the Graffiti program. We are in debt to the anonymous reviewers, whose positive and constructive criticisms greatly enhanced this paper. This work is supported by EPSRC grants GR/M45030 and GR/K/65706.

References

[1] R Bagai, V Shanbhogue, J Żytkow, and S Chou. Automatic theorem generation in plane geometry. In *LNAI, 689*. Springer Verlag, 1993.

[2] D Bailey. Finding new mathematical identities via numerical computations. *ACM SIGNUM*, 33(1):17 – 22, 1998.

[3] F Bennett, J Koza, M Keane, and D Andre. Genetic programming: Biologically inspired computation that exhibits creativity in solving non-trivial problems. In *Proceedings of the Symposium on AI and Scientific Creativity, AISB'99, Edinburgh*, 1999.

[4] S Chou. *Mechanical Theorem Proving*. D. Reidel, 1984.

[5] S Colton. Refactorable numbers - a machine invention. *Journal of Integer Sequences*, 2, 1999.

[6] S Colton, A Bundy, and T Walsh. HR: Automatic concept formation in pure mathematics. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1999.

[7] R Davis and D Lenat. *Knowledge-Based Systems in AI*. McGraw-Hill Advanced Computer Science Series, 1982.

[8] S Epstein. *Knowledge Representation in Mathematics: A Case Study in Graph Theory*. PhD thesis, Department of Computer Science, Rutgers University, 1983.

[9] S Epstein. On the discovery of mathematical theorems. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 1987.

[10] S Epstein. Learning and discovery: One system's search for mathematical knowledge. *Computational Intelligence*, 4(1):42–53, 1988.

[11] S Fajtlowicz. On conjectures of Graffiti. *Discrete Mathematics 72*, 23:113–118, 1988.

[12] S Fajtlowicz. The writing on the wall. Unpublished preprint, available from http://math.uh.edu/~clarson/, 1999.

[13] K Haase. Discovery systems. In *Proceedings of the European Conference on Artificial Intelligence*, 1986.

[14] R Hilderman and H Hamilton. Heuristics for ranking the interestingness of discovered knowledge. In *Proceedings of Methodologies for Knowledge Discovery and Data Mining, PAKDD-99, LNAI, 1574*. Springer Verlag, 1999.

[15] R Kennedy and C Cooper. Tau numbers, natural density and Hardy and Wright's theorem 437. *International Journal of Mathematics and Mathematical Sciences*, 13, 1990.

[16] Y Kodratoff and C Nédellec. Machine learning and comprehensibility. In *IJCAI'95 Workshop on Machine Learning and Comprehensibility*, 1995.

[17] T Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, 1970.

[18] P Langley. The computer-aided discovery of scientific knowledge. In *Proceedings of the first international conference on discovery science*, 1998.

[19] P Langley, H Simon, G Bradshaw, and J Żytkow. *Scientific Discovery - Computational Explorations of the Creative Processes*. MIT Press, 1987.

[20] C Larson. Intelligent machinery and discovery in mathematics. Unpublished preprint, available from http://math.uh.edu/~clarson/, 1999.

[21] D Lenat. *AM: An artificial intelligence approach to discovery in mathematics*. PhD thesis, Stanford University, 1976.

[22] D Lenat. Eurisko: A program which learns new heuristics and domain concepts. *Artificial Intelligence*, 21, 1983.

[23] D Lenat and J Brown. Why AM and EURISKO appear to work. *Artificial Intelligence*, 23, 1984.

[24] W McCune. Otter user guide. Technical Report ANL/90/9, Argonne National Laboratory, 1990.

[25] W McCune. Automated discovery of new axiomatizations of the left group and right group calculi. *Journal of Automated Reasoning*, 10(1):1–13, 1992.

[26] W McCune. A Davis-Putnam program and its application to finite first-order model search. Technical Report ANL/MCS-TM-194, Argonne National Laboratory, 1994.

[27] G Miller. On the $n^{log_2 n}$ isomorphism technique: A preliminary report. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 51 – 58, 1978.

[28] E Morales. DC: a system for the discovery of mathematical conjectures. Master's thesis, University of Edinburgh, 1985.

[29] T Nickles, editor. *Scientific Discovery: Case Studies*. D. Reidel, 1980.

[30] H Pistori and J Wainer. Automatic theory formation in graph theory. In *Argentine Symposium on Artificial Intelligence*, pages 131 – 140, 1999.

[31] G Ritchie and F Hanna. AM: A case study in methodology. *Artificial Intelligence*, 23, 1984.

[32] W Shen. Functional transformations in AI discovery systems. Technical Report CMU-CS-87-117, Computer Science Department, CMU, 1987.

[33] M Sims and J Bresina. Discovering mathematical operator definitions. In *Machine Learning: Proceedings of the 6th International Conference*. Morgan Kaufmann, 1989.

[34] S Singh. *Fermat's Last Theorem*. 4th Estate Limited, 1997.

[35] D Swanson. An interactive system for finding complementary literatures: A stimulus to scientific discovery. *Artificial Intelligence*, 91(2), 1997.

[36] R Valdés-Pérez. Machine discovery in chemistry: New results. *Artificial Intelligence*, 74:191–201, 1995.

[37] R Valdés-Pérez. Principles of human computer collaboration for knowledge discovery in science. *Artificial Intelligence*, 107(2):335 – 346, 1999.

[38] G Williams. Evolutionary hot spots data mining - an architecture for exploring for interesting discoveries. In *Proceedings of Methodologies for Knowledge Discovery and Data Mining, PAKDD-99, LNAI, 1574*. Springer Verlag, 1999.

[39] J Zhang. MCS: Model-based conjecture searching. In *Proceedings of CADE-16, LNAI, 1632*. Springer Verlag, 1999.